1

SSIS Solution Architecture

COPYRIGHTED MATERIAL

Imagine that this is the first day of a new internal or client project. You will have responsibility on the data and processing layer of the solution, which involves processing data — a lot of data — from several sources, and then either integrating systems, or maybe consolidating data for reporting. Maybe your background is a developer, or a database administrator (DBA), or a data designer, and you know SSIS fairly well. But now they are calling you the "data and ETL architect."

> *ETL is the acronym for Extraction, Transformation, and Loading, which is used to describe the data-processing layer in a data-integration or data warehouse solution.*

The project manager approaches you and says that the Vice President of Technology has asked the team to give him an estimate of the infrastructure needed. Furthermore, the business owner wants a high-level overview of how the solution architecture will help the company achieve the business need most efficiently. The project manager also wants your thoughts on the best way to approach the solution, how the development should be coordinated between team members, and how deployment should be handled.

Three meetings are scheduled for tomorrow to address these things, and you've been asked to provide your feedback in those areas.

Where do you start? How should you approach the solution design with SSIS as the main technology? How should all the pieces work together?

This chapter examines how to lay the foundation for successful solutions based on SQL Server Integration Services (SSIS). And, in fact, this whole book is about SSIS solutions to real-world requirements and challenges. It addresses questions such as the following:

- What are the problems and issues?
- What are the design principles?
- How do you put it all together for a complete and successful solution?

Before you dive into the technical challenges of a project, you must first step back and ensure that you are laying the right foundation. Jumping right in is tempting! But resist the urge, because you want to (and need to) set the precedence and patterns for the solution upfront. If you don't, chances are you won't be able to go back and make changes down the road.

As with all chapters in this book, this chapter is organized into the following three major sections:

- *"Problem"* — Coordinating and architecting an SSIS solution is not an easy task. The "Problem" section reveals some common challenges and common mistakes when planning and extending your ETL environment.
- *"Design"* — The "Design" section in this chapter examines the right approach to a project, and the long-term project aspects that you should set in motion early in a project.
- *"Solution"* — In many ways, the remainder of this book provides you with the solutions to make it all work together. This section launches you into the rest of the book, and shows how you can follow the chapters to build or redesign your SSIS solution.

Problem

Data and ETL projects have many challenges. Some challenges relate to data, some to enterprise integration, some to project coordination, and some to general expectations. This section begins by looking at the bigger picture of data within an organization, but then quickly looks at ETL projects and SSIS packages and execution.

Macro Challenge: Enterprise Data Spaghetti

Maybe your SSIS project is only a small solution in a bigger enterprise pond. The problem is that it can still cause a ripple effect when you tie it into your environment. Or, you can have challenges caused by an unwieldy enterprise environment when you try to implement your solution.

Figure 1-1 shows a not-so-nice telephone/electricity pole that illustrates the data nervous system of many organizations.

Figure 1-1

The problem with Figure 1-1 is that this mess didn't happen overnight! It grew into this twisted unorganized process because of poor planning, coordination, and execution. However, be aware that, a lot of the time, a corporation's politics may lead to this type of situation. Departments hire their own technical people and try to go around IT. Systems don't talk to each other nicely. Project pressures (such as time and budget) cause designers to cut corners.

Following are a few reasons why this kind of tangled mess happens in an organization's data processing, and examples of the many problems that this "unorganization" causes:

- *Debilitating dependency chains* — The biggest problem is that often systems are built on top of systems on top of systems. The core source data has been connected through so many precedence links that it takes more time and administrative and development overhead. Systems at the source and in the middle of dependencies become un-replaceable because of the amount of effort that switching to a new system would take.

- *Unknown and uncontrollable data processing* — The operations that process data are not centralized, and, in many cases, unknown because of department applications that are created without coordination with other applications. Processes run at uncontrolled times, and may impact systems within the processes even during peak times, which affects work efficiency.
- *Fragile enterprise changes* — Changes to applications are difficult and costly. They may break processes, or cause data integration or reporting applications to be inaccurate.
- *Delayed data access* — Even when the processes are somewhat controlled, the complicated system dependencies cause delays in data availability and nightly overhead processes that often run into mid-morning schedules. When they break, customer perception and employee efficiency are affected.

The "Design" section later in this chapter discusses how to approach your SSIS-based ETL project in the right way, and ensure that you are helping to solve the problem, rather than adding to it.

Micro Challenge: Data-Processing Confusion

Another common problem with data processing is when the logic contained to process data is overly complicated and confusing. Just like the macro enterprise problem, this problem usually is the result of changes over time where logic is modified and appended. It usually comes in one of two ways:

- *Runaway stored procedures* — Procedures that run with complicated logic and lots of temporary tables, inserts, updates, and/or deletes can be difficult to manage, and are often inefficient. Supporting the procedures is also very difficult because the logic is difficult to follow, and, many times, the developers or DBAs who wrote the code are unavailable. Overall, this type of process requires a lot of administration and wasted time spent on following and learning the process.
- *Unmanageable packages* — SSIS packages can also be designed with difficult-to-follow logic and sometimes complex precedence with hundreds of components used in a single package. These kinds of packages have challenges similar to those of runaway stored procedures, such as troubleshooting and the learning curve required for the process. Figure 1-2 shows the control flow of a package that has too many components to effectively manage. (The SSIS designer is zoomed in at 50 percent to fit on the screen.)

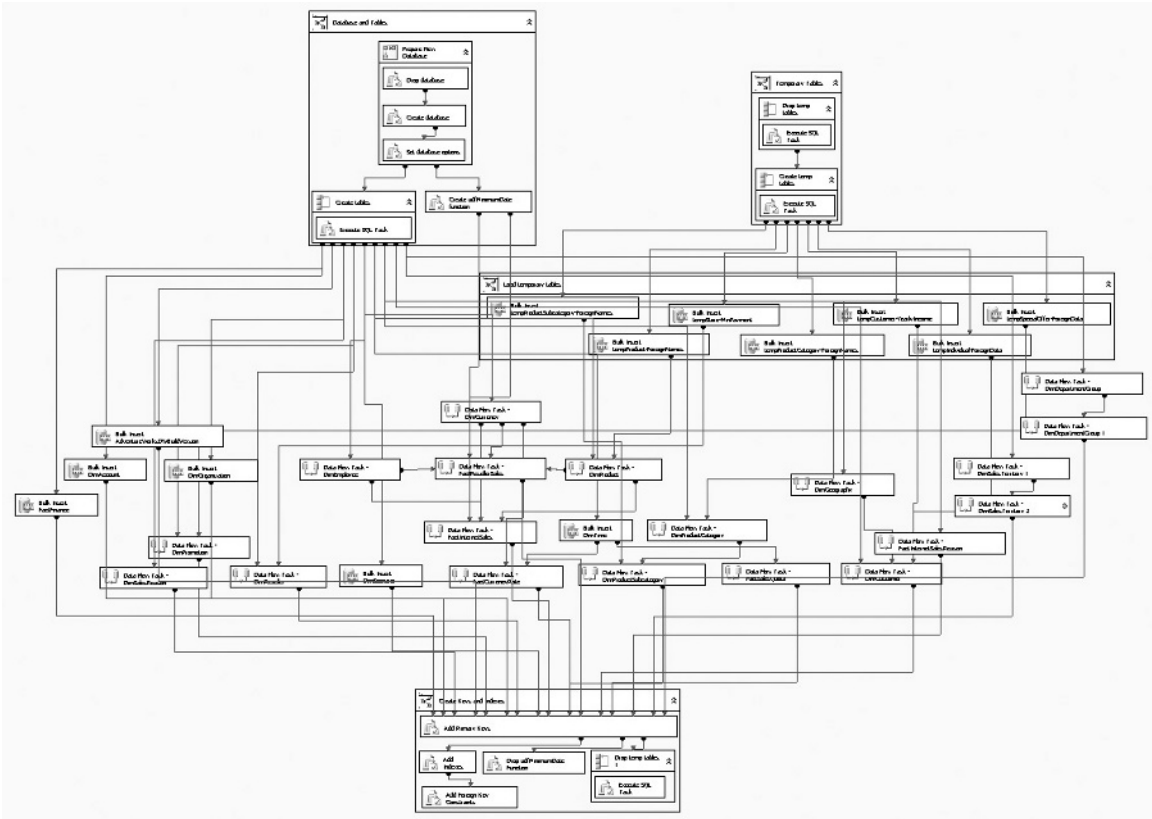

Figure 1-2

The overly complex control flow shown in Figure 1-2 is similar to an overly complex data flow, where too many components are used, thus making the development, troubleshooting, and support difficult to manage. The "Design" section later in this chapter proposes a better approach for SSIS packages called the *modular package approach*.

In summary, both of these types of processes (runaway procedures and unmanageable packages) are very difficult to support, and not suited to team development, error handling, and scalability (all of which are addressed in Chapter 12).

Problems with Execution and Troubleshooting

A couple of other issues that often come up in an ETL or data-integration solution are poor process coordination and difficulty doing root cause analysis. If the "what happened?" question can't be answered quickly and with confidence, then likely there is a problem with the overall solution execution and logging strategy.

Figure 1-3 shows the command-line output of an example SSIS package execution.

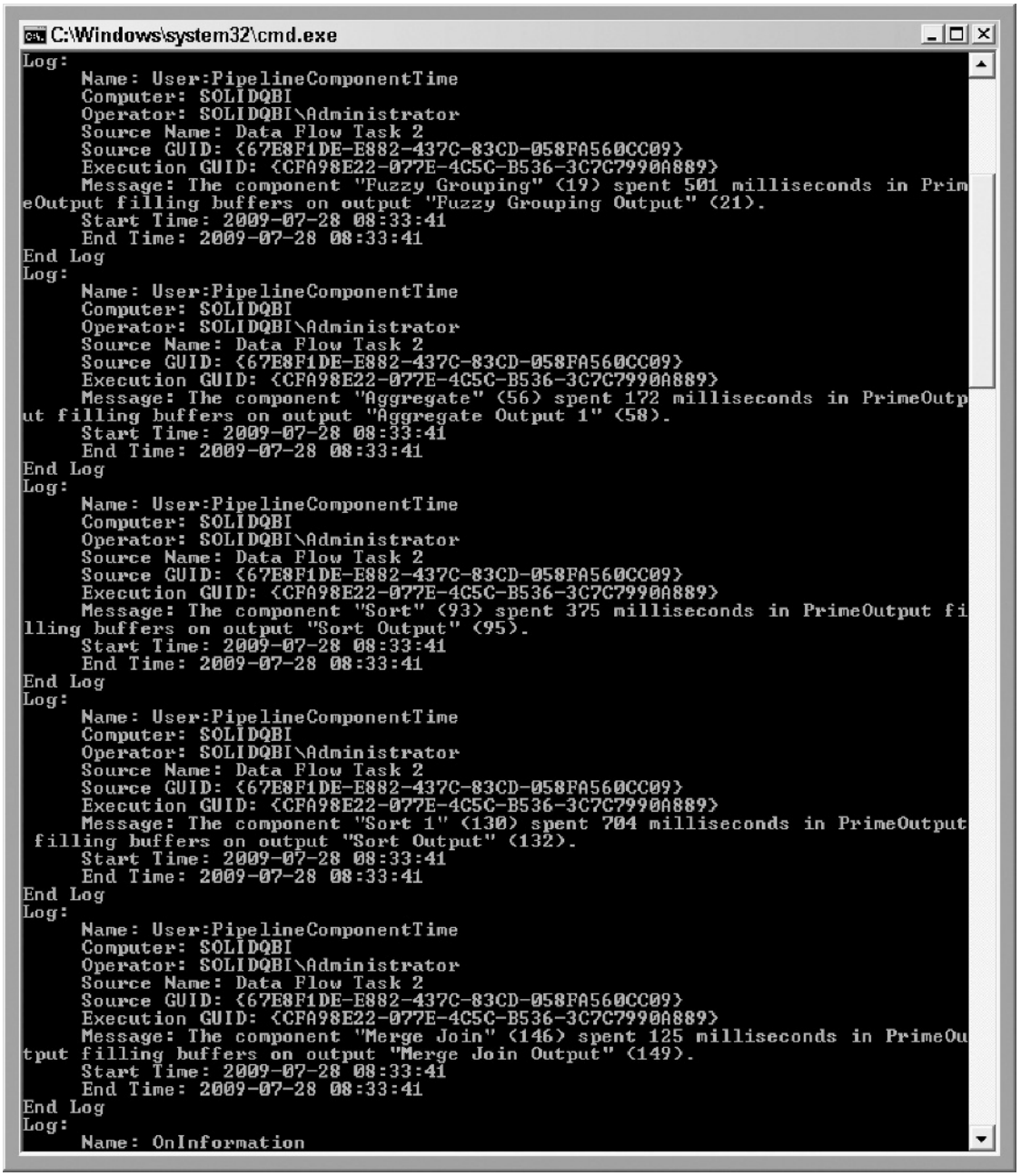
C:\Windows\system32\cmd.exe
Log:
Name: User:PipelineComponentTime
Computer: SOLIDQBI
Operator: SOLIDQBI\Administrator
Source Name: Data Flow Task 2
Source GUID: {67E8F1DE-E882-437C-83CD-058FA560CC09}
Execution GUID: {CFA98E22-077E-4C5C-B536-3C7C7990A889}
Message: The component "Fuzzy Grouping" (19) spent 501 milliseconds in PrimeOutput filling buffers on output "Fuzzy Grouping Output" (21).
Start Time: 2009-07-28 08:33:41
End Time: 2009-07-28 08:33:41
End Log
Log:
Name: User:PipelineComponentTime
Computer: SOLIDQBI
Operator: SOLIDQBI\Administrator
Source Name: Data Flow Task 2
Source GUID: {67E8F1DE-E882-437C-83CD-058FA560CC09}
Execution GUID: {CFA98E22-077E-4C5C-B536-3C7C7990A889}
Message: The component "Aggregate" (56) spent 172 milliseconds in PrimeOutput filling buffers on output "Aggregate Output 1" (58).
Start Time: 2009-07-28 08:33:41
End Time: 2009-07-28 08:33:41
End Log
Log:
Name: User:PipelineComponentTime
Computer: SOLIDQBI
Operator: SOLIDQBI\Administrator
Source Name: Data Flow Task 2
Source GUID: {67E8F1DE-E882-437C-83CD-058FA560CC09}
Execution GUID: {CFA98E22-077E-4C5C-B536-3C7C7990A889}
Message: The component "Sort" (93) spent 375 milliseconds in PrimeOutput filling buffers on output "Sort Output" (95).
Start Time: 2009-07-28 08:33:41
End Time: 2009-07-28 08:33:41
End Log
Log:
Name: User:PipelineComponentTime
Computer: SOLIDQBI
Operator: SOLIDQBI\Administrator
Source Name: Data Flow Task 2
Source GUID: {67E8F1DE-E882-437C-83CD-058FA560CC09}
Execution GUID: {CFA98E22-077E-4C5C-B536-3C7C7990A889}
Message: The component "Sort 1" (130) spent 704 milliseconds in PrimeOutput filling buffers on output "Sort Output" (132).
Start Time: 2009-07-28 08:33:41
End Time: 2009-07-28 08:33:41
End Log
Log:
Name: User:PipelineComponentTime
Computer: SOLIDQBI
Operator: SOLIDQBI\Administrator
Source Name: Data Flow Task 2
Source GUID: {67E8F1DE-E882-437C-83CD-058FA560CC09}
Execution GUID: {CFA98E22-077E-4C5C-B536-3C7C7990A889}
Message: The component "Merge Join" (146) spent 125 milliseconds in PrimeOutput filling buffers on output "Merge Join Output" (149).
Start Time: 2009-07-28 08:33:41
End Time: 2009-07-28 08:33:41
End Log
Log:
Name: OnInformation

Figure 1-3

If you were to consider spending time trying to work through this output when trying to figure out what went wrong, then you should consider implementing a better execution and auditing structure. This includes package execution in your development environment.

If you have just turned on the out-of-the-box SSIS logging and are capturing results to output to a table, it still may not be enough. If you write custom queries every time against the SSIS logging table to figure out what happened, then you also need a better strategy.

Infrastructure Challenges

When designing an SSIS ETL solution, how do you determine the infrastructure components such as server hardware, memory, processor cores, network switches, disk arrays, storage networks, and I/O controllers? Related to that, where should you run your SSIS packages taking into consideration sources, destinations, and other applications, while balancing hardware scalability and location within your network topology?

These questions are not trivial, and the answers depend on a lot of factors, including processing windows, source and destination availability, application impact and availability, network bandwidth, fault-tolerance requirements, and so on.

I/O is usually the biggest bottleneck, and the one most often overlooked. *I/O* (or, more precisely, *disk I/O*) is the throughput that the system can handle on the drive volumes. And this challenge is not just about trying to get the greatest throughput on a single drive. You must consider staging and temporary environments, logging, and current and historical data. And you must balance it all with hardware availability and budget.

The reason disk I/O is so important when considering a data-integration or ETL effort is because of the nature of what you are doing, including the following:

- *Bulk operations* — ETL and integration efforts typically process data in bulk, meaning that, when a process is kicked off (hourly, daily, or weekly), data is moved from sources to destinations with some transformation processes in-between. The processes usually move or integrate thousands or millions of records. That can be a lot of data that moves between systems, and it generates a lot of disk activity.
- *Source databases* — Processes that are extracting a lot of data from sources incur disk overhead either by the sheer volume of the data, or when complicated queries against large tables require temporary operations that use disks (such as the SQL Server `TempDB`).
- *Destination databases* — The nature of relational databases requires that data be stored to disk before a transaction is marked as complete. When inserting or updating a lot of data, the server must wait until the data is committed for the process to be complete.
- *Staging databases* — Staging databases and tables are a common intermediate step in an ETL process and can be used for many reasons. Any time that data is landed to a staging database and then extracted from the staging database, it has the overhead of both the insert and the select, which can, at times, be done simultaneously with inserts into the destination database and, therefore, the I/O is compounded.
- *File management* — A lot of ETL operations deal with files such as delimited files, XML, EDI, and so on. Large files require file management, archiving, and sometimes processing, and, therefore, incur disk I/O overhead.

The bottom line is that you will most likely have a disk I/O bottleneck in your data-processing operations, and you'll need to plan and manage for that to meet your service level agreements (SLAs) and performance requirements.

Other Challenges

The list of common project challenges can go on and on, but here are a few more:

- *Data challenges* — Of course, you will have data challenges in your project anywhere from missing records to dirty data to bad data, and you will need to understand those problems as soon as possible so that you can set the expectations upfront about what can and what cannot be done about them. Although you can do a lot in SSIS (including fuzzy matching), magic is not on the component list — you can't make up data that doesn't exist. Don't overpromise. Be realistic.
- *Corporate political challenges* — This book is about solving problems with a technology, namely SSIS. But, because you are trying to solve problems, you are going to be dealing with people. Everyone has an agenda, and a lot of times those agendas will not be in your project's best interest. Watch out for people who are threatened because you are changing the way things are done (even when it is for the better), or because your solution will be replacing one of their legacy applications, or because they are territorial about their "sandbox." You want to fix the enterprise spaghetti shown in Figure 1-1, but don't forget that some people have their tribal knowledge and make their living by keeping the corporate data nervous system tangled.
- *Requirement and scope challenges* — Any project has scope creep. Just be careful about how the changes affect the project timeline, and don't leave data validation until the last day. You'll get burned!

Design

Now that you are scared, step back and take a deep breath. Designing an ETL process is doable, and, with the right approach in mind, you can be successful. This section discusses the overall design approach to an SSIS-based solution by examining the following:

- Choosing the right tool
- Solution architecture
- Project planning
- Package design patterns
- Server and storage hardware
- Package execution location

Choosing the Right Tool

This book is about applying SSIS. You are probably reading it because you assume that SSIS is the right tool for the job. That's probably the case. However, be sure to consider what you are doing, and ensure that using SSIS is in line with what you are doing.

Think about all the different types of data-processing needs that you have across your organization:

- Data synchronization between systems
- Data extraction from enterprise resource planning (ERP) systems
- Ad hoc reporting
- Replication (both homogeneous and heterogeneous)
- PDA data synchronization
- Legacy system integration
- Data warehouse ETL processing
- Vendors and partner data files integration
- Line-of-business data processing
- Customer and employee directory synchronization

As you may know, when it comes to data processing, a lot of tools are out there. Some are created for specific situations (such as folder synchronizing tools), whereas other tools are designed to perform a variety of functions for different situations. So, the traditional question often posed is which tool can best meet the business and logical requirements to perform the tasks needed?

Consider the host of tools found in the ever-evolving Microsoft toolset. You can use Transact SQL (TSQL) to hand-code a data load, Host Integration Server to communicate with a heterogeneous data source, BizTalk to orchestrate messages in a transactional manner, or SSIS to load data in batches. Each of these tools plays a role in the data world.

Although overlaps exist, each tool has a distinct focus and target purpose. When you become comfortable with a technology, there's always the tendency to want to apply that technology beyond its intended "sweet spot" when another tool would be better for the job. You've no doubt heard the phrase "when you're a hammer, everything looks like a nail." For example, C# developers may want to build an application to do something that SSIS could potentially do in an hour of development time. The challenge everyone faces entails time and capacity. There is no way everyone can be an expert across the board. Therefore, developers and administrators alike should be diligent about performing research on tools and technologies that complement each other, based on different situations.

For example, many organizations use BizTalk for a host of purposes beyond the handling of business-to-business communication and process workflow automation. These same organizations may be perplexed as to why BizTalk doesn't scale to meet the needs of the organization's terabyte data warehousing ETL. The easy answer is that the right tool for bulk Business Intelligence (BI) processing is an ETL tool such as SSIS.

Be Careful About Tool Selection

In some client environments, an ETL tool may be chosen without consideration for the availability of industry skills, support, or even the learning curve. Even though the tool could perform "magic," it usually doesn't come with a pocket magician — just the magic of emptying your corporate wallet. In many cases, thousands of dollars have been spent on an ETL tool that takes too long to master, implement, and support. Beyond the standard functionality questions you should ask about a tool, be sure to also consider the following:

- ❑ Your internal skill sets
- ❑ The trend of industry use of the tool
- ❑ How easy it is to learn
- ❑ The ease of supporting the tool

Overall Solution Architecture

The reality is that creating a perfectly efficient enterprise data ecosystem is impossible. But there are certainly levels of efficiency that can be gained when your SSIS solution is planned and implemented thoughtfully. Figure 1-4 contrasts Figure 1-1 by showing a city's central power station, organized and efficient.

Figure 1-4

The tendency when developing a new integration or ETL system is to get it done as quickly as possible. What often happens is that the overall architecture is not integrated well into an organization's environment. Maybe some time is saved (and that is even questionable), but in the end, more time and money will be wasted.

A solution architecture should have several key data-processing objectives. The following apply to SSIS-based solutions, but also relate generically to any data-processing solution architecture:

- The solution should coordinate with other data-centric solutions in the enterprise. Do not build a separate data silo, especially if your effort is a data warehouse or data mart — that causes multiple versions and variations of the data.
- Source data that is required for the solution must be extracted as close to the source as possible and not plugged at the end of a long dependency chain. (Be sure to follow the previous bullet point).
- The solution should have a centralized administration and execution strategy that coordinates with other systems, or follows the practices of other corporate systems. This does not require limiting a scale-out architecture, but simply that the support structures are centralized.
- Real-time execution auditing is also needed to know what is happening and what did happen. This information will go a long way in supporting a system. In addition, you should have a way to track data back to the source. This tracking is critical for both data validation and troubleshooting data errors.
- The processes must have rollback layers. In other words, if your requirements necessitate a complicated or long process, don't require the whole process to be re-run from the start if the last part breaks. Plan for restarting at interim points after the issues are identified. Doing so also enables you to easily compartmentalize changes in the ETL solution.

These objectives represent the larger picture of an overall solution architecture. Other aspects, of course, are important and situational to what you are building in SSIS. Two common types of data-processing efforts are discussed in the following sections: system integration and data warehouse ETL. These fit well into the SSIS toolset.

Data Integration or Consolidation

One common use of SSIS is to integrate data between systems, or to synchronize data. For example, you may want to create a business-to-business portal site, and you may need the site to interface with the source data on the mainframe. In this case, you may get the data delivered in nightly extracts from the mainframe and load it into your SQL Server table.

Another very common ETL task that DBAs face is receiving files from File Transfer Protocol (FTP) servers (or on network shares) that must be processed and loaded into another system. This type of process involves moving files, and then processing the data, which may involve de-duping (removing duplicates), combining files, cleaning bad data, and so on. Two systems may also need to talk to one another or pass business keys in order for records to be matched between environments.

Figure 1-5 shows an example solution architecture that integrates data between different systems in an enterprise.

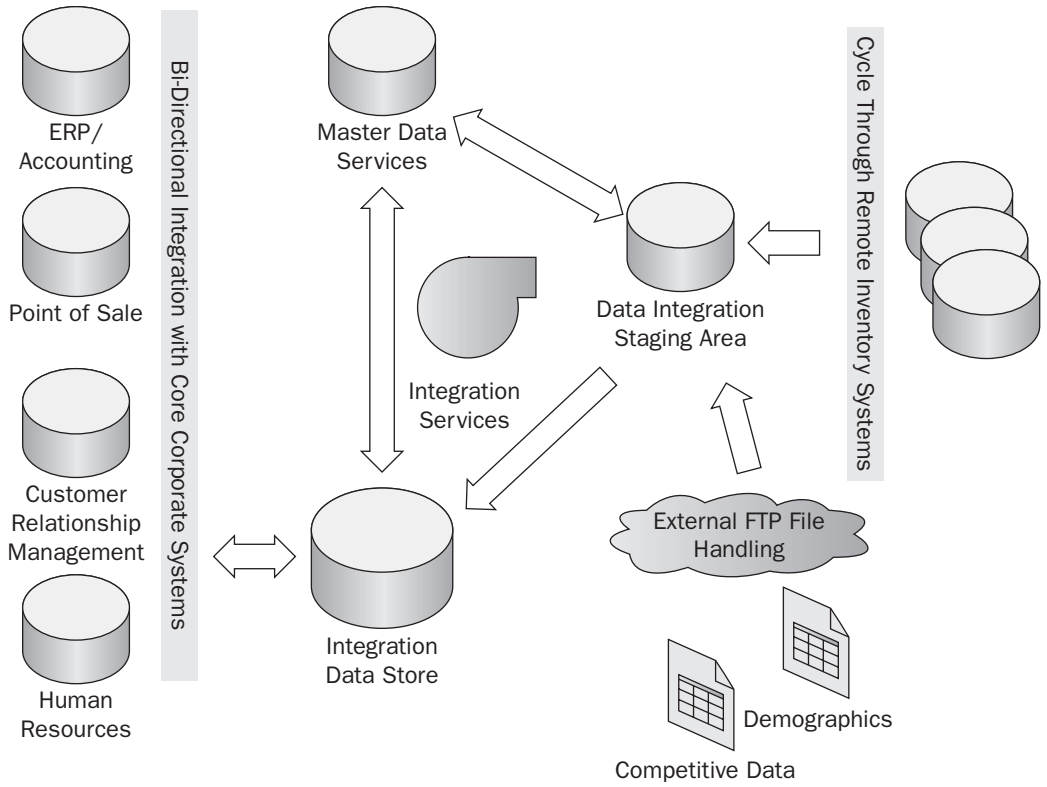

Figure 1-5

In this diagram, data from different systems is integrated. Often, a master data management service (sometimes called *master dimension management*) helps to synchronize entities between systems so that an organization's data relates (for example, so that a customer record from the customer relationship management, or CRM, system can tie into the ERP sales system). This data process contains some aspects that are bidirectional, and other parts that perform extraction and loads. Data staging is used in this example to help integrate the data, and a data store is used to centralize many of the corporate data processes (which helps alleviate the long chains of system dependencies).

Of course, other variations of system integration solutions exist, such as consolidation of different divisional data, especially when companies go through mergers and acquisitions.

Data Warehouse ETL

One of the more common uses of SSIS is for performing data warehouse ETL. Data warehousing focuses on *decision support*, or enabling better decision making through organized accessibility of information. As opposed to a *transactional system* such as a point of sale (POS), Human Resources (HR), or CRM that is designed to allow rapid transactions to capture information data, a data warehouse is tuned for reporting and analysis. Because data warehousing is focused on the extraction, consolidation, and

reporting of information to show trending and data summaries, the ETL part of a data warehouse is important and critical.

Processing ETL for data warehousing involves *extracting* data from source systems or files, performing *transformation* logic on the data (to correlate, cleanse, and consolidate), and then *loading* a data warehouse environment (for reporting and analysis). Figure 1-6 shows common data-processing architecture for a data warehouse ETL system.

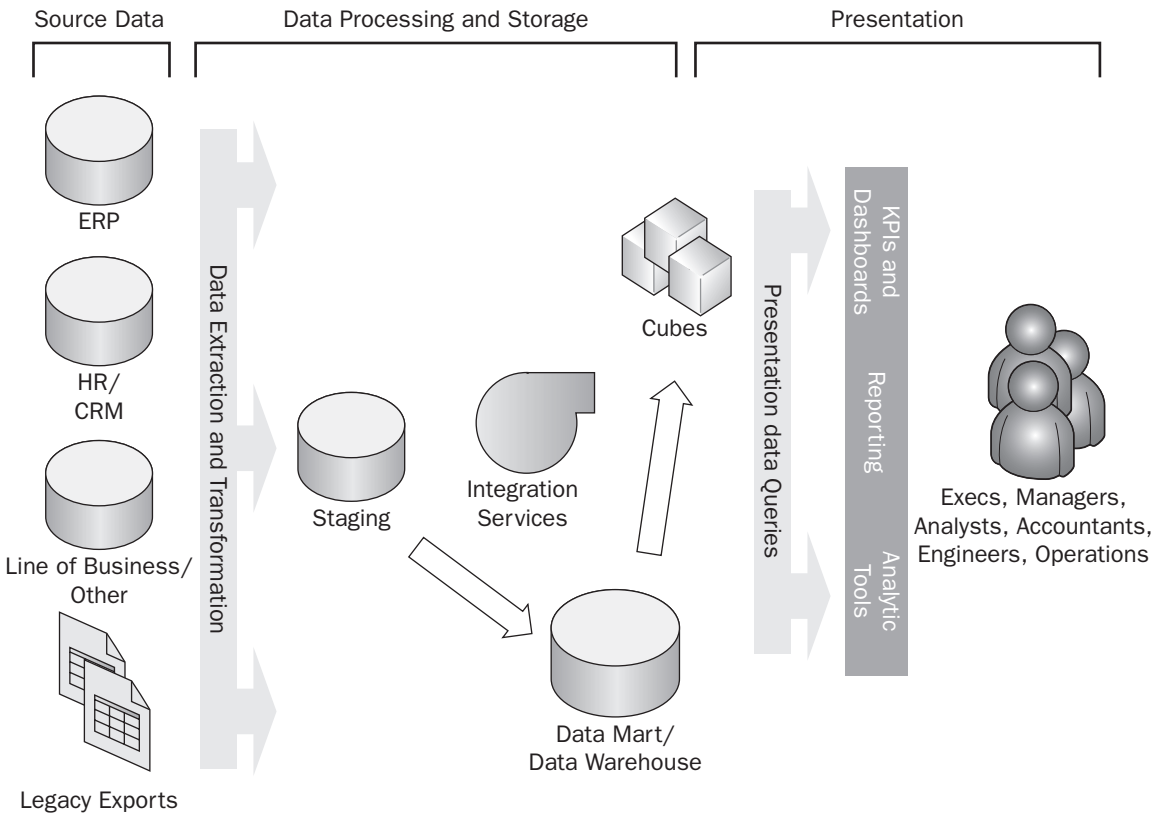

Figure 1-6

For those who are already versed in ETL concepts and practice, you may know that when it comes to developing a data warehouse ETL system, moving from theory to practice often presents the biggest challenge. Did you know that ETL typically takes up between 50 and 70 percent of a data warehousing project? That is quite a daunting statistic. What it means is that even though presenting the data is the end goal and the driving force for business, the largest portion of developing a data warehouse is spent not on the presentation and organization of the data, but rather on the behind-the-scenes processing to get the data ready.

Project Planning and Team Development

This is not a project methodology book, but you should give some thought to your solution approach. Whether your overall objective is system integration or warehouse ETL, you should give consideration to using an agile development methodology. An *agile methodology* is an iterative approach to development. You add features of the solution through smaller development cycles, and refine requirements as the solution progresses.

Agile Benefits

Even if your solution does not involve a user interface (such as a system integration), an agile approach enables you to tackle aspects of the solution in smaller development cycles, and to troubleshoot data issues along the way. Following are some the general benefits of this approach:

- *Project and task definition* — The agile approach requires definition of the tasks in a project and the prioritization of the tasks, which are set and known by the technology team and the business or project owners. Tasks can change as a better understanding of the requirements is defined.
- *Scope management* — Given the clear definition of the activities in an iteration, the scope is set, and any deviation is known to the parties involved, and agreed upon. In essence, project communication is clearer for all parties — developer, management, and ownership.
- *Addressing of issues and obstacles* — Part of the ongoing process involves identifying the areas in the project that can impede progress. These are highlighted and addressed soon in the process.
- *Roles and responsibility clarity* — Roles are clearly defined and tasks are assigned, which limits the possibility of team members spinning their wheels in the process.

Agile Cautions and Planning

However, you must exercise some caution. Do not use an agile methodology to foster bad architecture practices. In other words, if you are just using the agile approach and you or your developers' ultimate goal is to meet the tasks and deadlines in whatever way possible, you are going to fall into the trap of compounding the problems of your overall data nervous system (that is, those problems outlined in the earlier "Problem" section).

You must ensure that you have an overall solution architecture, and your agile tasks must fit in that plan and support the cause.

Therefore, whatever project methodology you use, be sure to push for an upfront plan and architecture. If you don't, you will likely run into the tyranny-of-the-urgent problem — that means that you will get overwhelmed with the tasks and, as you throw them together, your solution gets out of control, code is messy, and your stress will be compounded over time.

Following are a few things to consider in your development process:

- Plan for an early-on proof-of-concept, and use the proof-of-concept to iron out your data process architecture.
- Set your SSIS package and database conventions upfront, including your auditing structure (as discussed in Chapter 2).

- ❑ Estimate your data volumes in one of the initial development cycles so that you can purchase the right hardware.
- ❑ Get your server storage ironed out upfront. Be sure to set expectations with the storage group or vendor early on in the process.
- ❑ Plan out your package storage and deployment plan in one of the initial phases. (Chapter 3 provides an in-depth discussion of this topic).
- ❑ In every development cycle, be sure to include a data-validation task so that you can have data checkpoints along the way, rather than having one data-validation test at the end (which often leads to changes).
- ❑ In regard to SSIS data-related solutions, you must plan upfront any initial data load requirements. If you leave out this planning step, you will likely underestimate the overall solution scope.

Data Element Documentation

Not many developers or system architects are fans of documentation — or at least writing documentation. However, it is a necessary task in any data-centric or ETL project.

Again, this book is more about SSIS solutions than project management, but given the importance of tracking data, included here are some recommendations on data-tracking documentation that can help you in your project.

Data documentation is about tracking the source and destination data elements, data profiling, and mapping the data elements to requirements. You must be diligent about these tasks, and keep them up-to-date, because doing so can help you keep control of the data your project uses. Documentation is also useful in future administration and lineage.

The following data-tracking documents should be used above and beyond your usual documentation (requirements, conceptual design, physical design, ETL design, and so on).

Source Data Dictionary, Profile, and Usage

The source data dictionary is about more than just a definition of what the data elements represent. It's also about what the data looks like, and what destination elements it is used in. Planning sessions can then refer to the source dictionary to help validate requirements and data availability.

You should structure this in two sections: entity definitions and element definitions.

Table 1-1 provides some details for entity tracking.

Table 1-1

Item	Description
Table or filename	This names the file or table and any ongoing naming conventions (such as name variations if different systems are involved, or if files have timestamps included).
Source and definition	Describes the source system where the data originates, and general data that the file contains.
Number of initial records and size	If the solution includes an initial data load, this represents the number of records that are included in the initial data, and the size of the file or table.
Number of incremental records and size	For ongoing data loads, this describes how many records are involved in the incremental source data, and the size of the incremental file or table.
Entity usage	How the source table or file is used in the solution.

Table 1-2 provides some details for element tracking.

Table 1-2

Item	Description
Source table or file	The table of the file that the element is sourced from.
Source column or field	Name of the table column of field from the file.
Definition	Describes the usage of the element in the source.
Data profile analysis	An analysis of the data profile — completeness, value variations, dependencies on other elements, key usage, or uniqueness.
Element usage	Lists the destination tables and columns that this source element is used in, which will be important to keep up-to-date.

Destination Data Dictionary, Usage, and Mapping

Tracking the destination elements so that you can use them to understand what the elements are for, where they came from, and how they are used is also important.

The destination dictionary describes the elements, but also describes the mapping from the source. This is invaluable in the ETL process. Again, you should include both an entity mapping and an element mapping description.

Table 1-3 provides some details for entity tracking.

Table 1-3

Item	Description
Table name	This is the destination table name, schema, and database that the table is used in.
Table description	Describes the use of the table in the overall entity-relationship diagram (ERD), and what general records and grain are included in it.
Keys and grain	Lists the primary key and any candidate keys in the table, and the data grain of the table.
Number of initial records	This is the count of the number of expected rows in the table.
Yearly record growth	Estimates the number of additional rows that will be added to the table.
Source entity mapping	Lists the source tables of files that are involved in the population of the table.

Table 1-4 provides some details for element tracking.

Table 1-4

Item	Description
Destination table name	The table and schema that the column is in.
Destination column name	Name of the table column.
Column description	Describes the usage of the column within the source.
Data type description	Describes the expected data types and ranges used in the column.
Usage type	Describes the type of usage for the column, such as a primary key, candidate key, foreign key, auditing column, descriptor column, and so on.
Source mapping	Lists the source fields used to populate the column, and describes the detailed mapping and transformations needed from the source to the data elements in the column. This is crucial for ETL processing.

Just as a review, this discussion only addresses the tracking of data elements, and is supplementary to the overall solution documentation. You may have other related data documentation, or you may choose to include additional items in your documentation (such as partitioning strategy of the destination table, or other pertinent things about the source data availability or data processing).

Package Design Patterns

The way you design your packages is important for the team development, deployment, future changes, ongoing support, and maintenance. In the "Problem" section earlier in this chapter, an example package was shown that had too much logic in a single package. A better approach is available through the use of modular packages and master packages.

Modular Packages

Instead of putting a lot of your data processing in a single package, focus your packages so that the processing logic contained in them is manageable, and the precedence is not overly complicated. This is called *modular package development,* and it provides the following benefits:

- *Facilitates team development* — A team of developers can be working on different aspects of the solution at the same time in different SSIS packages. Also, a single modular package is easier to unit test.
- *Eases source control and versioning* — Smaller packages are easier to identify in a source control system, and versioning can isolate changes easier.
- *Simplifies deployment* — Deploying changes with modular packages allows only the logic groups that change to be deployed.
- *Allows multi-use package execution* — A package may be used in multiple execution groups for related package execution, which reduces redundancy.
- *Helps identify errors* — When you're troubleshooting, isolating package bugs and failures is easier with smaller packages.
- *Enables better auditing and support* — Administration of an ETL solution is easier to track and audit with smaller module packages that are enabled with auditing and logging.

What does a modular package look like? Package designs vary, depending on the solution and requirements. But a good general rule is to keep the components visually manageable in the package designer without requiring a lot of scrolling to find aspects of the solution.

Figure 1-7 shows a package control flow that demonstrates a modular package.

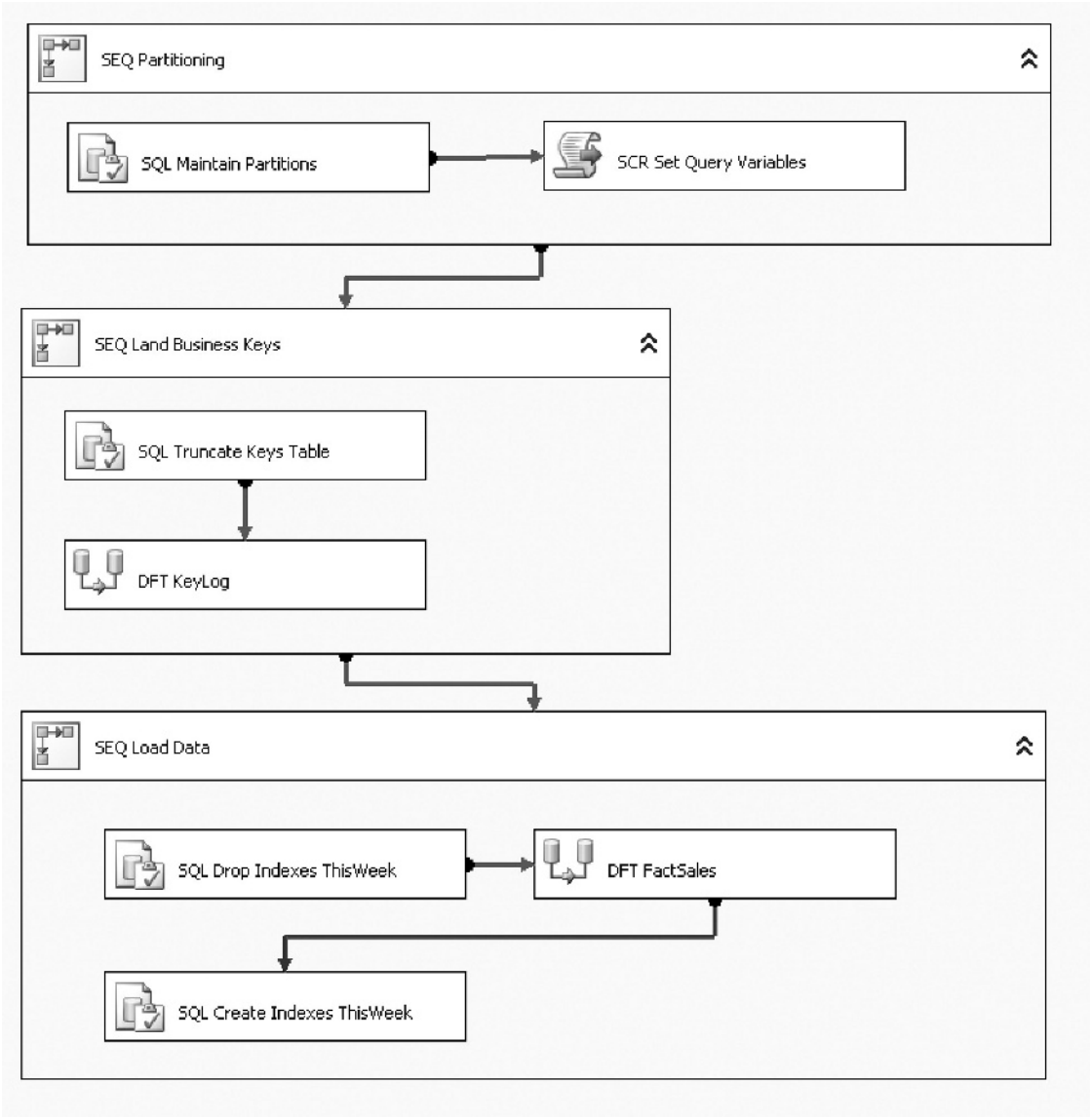

Figure 1-7

In this example, a couple of data flows and a few other tasks support the processing. In all, ten tasks are in the control flow, which is a very manageable group.

Master Packages

The way to still keep your complicated order of data processing (or precedence) is to coordinate the execution of the modular packages through a master package. A *master package* (or *parent package*) uses

the Execute Package Task to tie together the modular child packages so that they execute in the right order. Logging and auditing can be included to help facilitate an overall execution auditing and administrative support.

Figure 1-8 shows an example parent package.

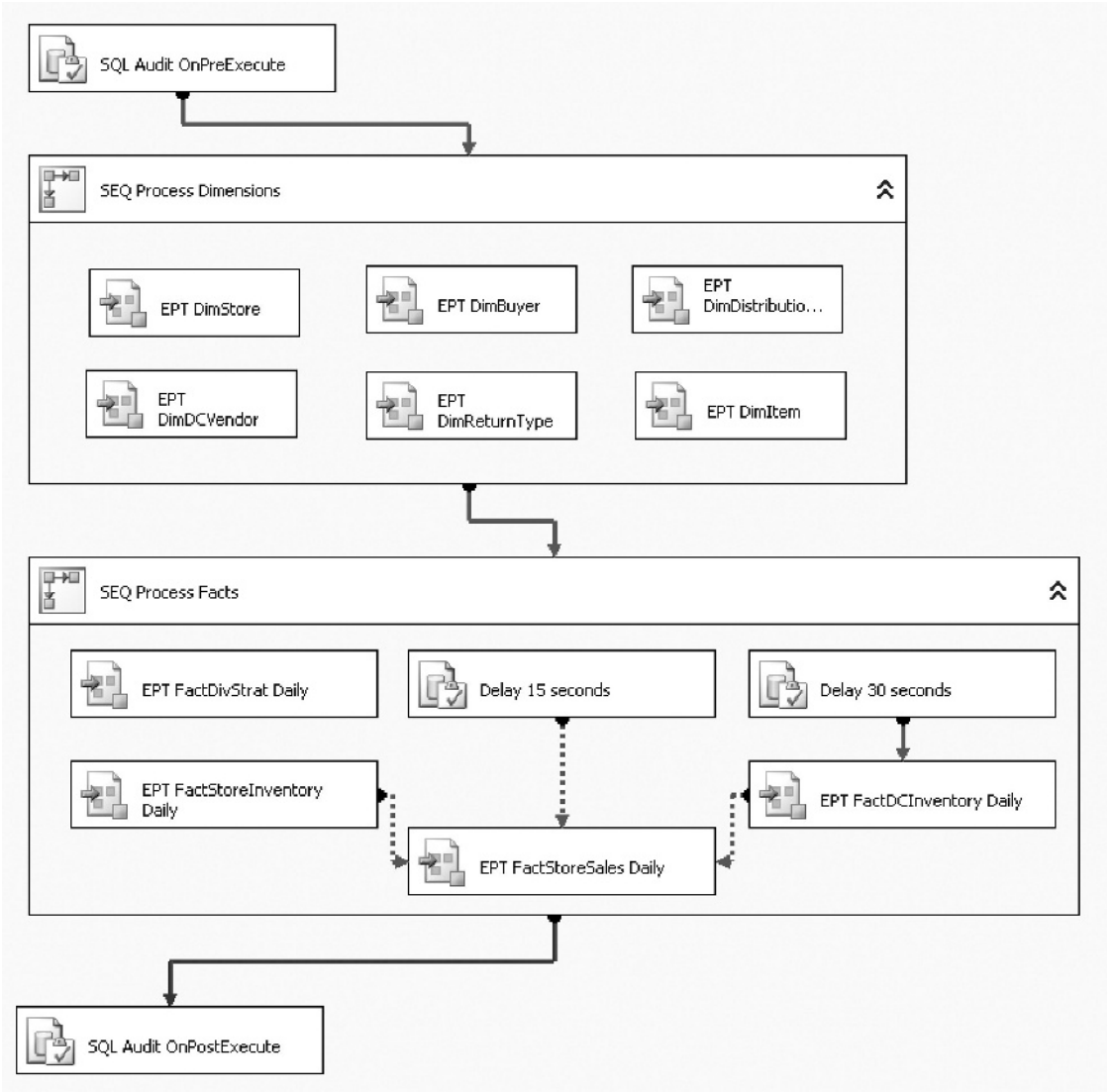

Figure 1-8

Each Execute Package Task ties to a package either stored in the file system, or in the `msdb` database package store. In many solutions, you will need to execute a set of packages at different times and with different precedence. The master package allows this, and helps implement a rollback and checkpoint system. Chapter 2 provides more coverage of this topic when discussing the building of a package framework.

Server and Storage Hardware

Identifying hardware upfront is a "Catch-22." In other words, you may not know the total scope of your project, or even what to expect from a load standpoint, but you are asked to estimate the hardware requirements for the project.

Server Hardware

Here are some general principles to follow concerning the server hardware:

- *64-bit* — You can't really purchase a server these days that is not an x64 processor architecture. This is good news. But be sure that the x64-bit version of Windows Server OS is installed, and that the x64 version of SQL Server 2008 is installed.
- *Multicore processors* — The biggest advantage to multicore processors (that is, dual core, quad core, and six core, as of the writing of this book) is that for SQL Server 2008, you are only paying per-socket. If you have a two-CPU quad core, the OS and SQL Server will see eight CPUs, but you are only buying the license for two sockets.
- *Memory* — Memory is relatively cheap and very valuable for SSIS. You should use at least 16GB of memory, but preferably 32GB or 64GB, or more, depending on the intensity of your ETL and how much you are utilizing the SSIS data flow.

You are now probably asking, "But how many cores and how many servers should I target?" There is no definitive answer to that question because it is so dependent on your ETL solution complexity, volume, and schedule.

If you must estimate, keep in mind that for a small-scale ETL solution that is dealing with less than 10GB of data (during your ETL), and that is working with a destination database that is less than 1TB, your ETL machine can probably run on a two-socket dual-core or quad-core server with 16GB or 32GB of RAM.

For a medium ETL solution (where your ETL requires execution in a smaller window, and your recurring volumes are larger and/or your destination database is in the multi-terabyte range), consider a quad socket with multicore processors, and at least 32GB of RAM or possibly 64GB with the option to expand.

Larger solutions are really dependent on so many factors, and you should also consider scale-out ETL (which is discussed in Chapter 10). Thus, recommending a general rule — especially if you are building an SSIS server that will run all your enterprise ETL operations — is difficult.

Again, there is so much context that will really drive the hardware requirements, that you must evaluate your situation and customize a recommendation on what you see in your solution.

Development and Test Servers

For mission-critical ETL processes, you must have a test environment that mirrors your production hardware. It will be costly, but consider the cost if you deploy a change that you couldn't test and your server goes down.

If your budget is restricted, and your ETL process is not mission-critical to your business, then your test environment can be a scaled-down version of your production servers or a virtual server. One option to save on cost is to use the same server for both development and testing, and then you may be able to use equivalent hardware. Use different database instances and packages for your testing.

One aspect of setting up test servers that is critical is that the number and naming of your volumes (drive letters) must match between all your environments. In other words, if you have `G:`, `H:`, and `I:` as logical drives on your production server, ensure that your development and test machines have the same logical drives, and the folder structures and databases are the same as well. You don't necessarily need to have the same number of physical drives, but you should partition what you do have into the same logical volumes. Doing so can alleviate a lot of deployment pain.

ETL Collocation

Sharing the source or destination server with your ETL process (called *collocation*) is an option if your source or destination is not impacted by your ETL process. What you must watch out for is the database engine taking all the available RAM on the server, and starving your SSIS process of memory, which can be very debilitating to the ETL.

You can configure the SQL Server memory through the `sp_configure` TSQL statement. You can begin your SSIS processes by limiting the RAM that SQL uses, then run your ETL, and at the end of the ETL process, reset the memory usage on the SQL Server.

SSIS collocation is best for smaller to medium solutions where the ETL only runs nightly or weekly. The next section clarifies the execution location of your environment.

Storage Hardware

As mentioned in the "Problem" section earlier in this chapter, the storage hardware is important because you are performing bulk operations. The more throughputs you can generate with your disk subsystem, the better.

How do you estimate hardware needs? Doing so is very difficult at the start of a solution, but if you can get a gut sense of the record volume and growth, then you can probably do it. A good DBA will be able to help estimate the table sizes by taking the estimated row width, multiplying that number by the expected rows, and then adding some overhead for indexing and growth. (You must consider a lot of factors such as, the SQL Server data page width and free space.)

Disk Volumes and Configuration

Following are some general principles to follow as you try to estimate your disk volumes:

- Limit the use of internal server storage, and especially don't put your databases on the boot/system drive (`C:`).
- Go with smaller, faster drives, and more of them, rather than bigger and slower drives (except for backups and archive). You can get a lot more throughput for the drives because you can stripe more drives.
- Separate your staging and `TempDB` databases on separate drives (even when using a storage area network, or SAN) because, for ETL operations, you will create a bad I/O bottleneck if your source or destinations share the same drives as staging.
- Logs should also be on a separate drive because, even if your database is set up as simple recovery (where the log file gets truncated), you will still generate a lot of log activity during ETL operations.
- If you estimate your destination database will be 1TB, you will probably need 3–4TB of raw drive space to accommodate for logs, temp, staging, disk striping, redundancy (RAID), and so on. Set this expectation upfront!

Storage Area Network (SAN) Versus Direct Attached Storage (DAS)

Both Storage Area Networks (SANs) and Direct Attached Storage (DAS) have their benefits and drawbacks. SANs come at a higher price, but have the benefit of adding better redundancy, caching, controller throughput, more drives in a stripe, fault tolerance (clusters in different cities), advanced disk mirroring (where a mirror can be split and mounted on other servers), dual read in a mirror (where both drives in a mirror can be read at the same time), and so on.

DAS has the benefit of cost (a fraction of the cost of a SAN), but can also achieve similar throughput (and, in some cases, faster throughput, but without the caching) and easier control of the setup and configuration.

For mission-critical ETL processes and databases, use a SAN. But if your solution doesn't need that high availability, you can consider DAS. Or, you can consider DAS for your staging environment, and SAN for your production databases.

> *This is an SSIS ETL book, so you should consult the current best practices out there for recommendations on drive configuration. However, just remember that ETL generates a lot of I/O in a short amount of time and, therefore, you should watch out for recommendations that are targeted for transactional systems.*
>
> *A lot of varying and seemingly contradictory recommendations are out there, but each is based on a set of assumptions for different types of data-centric solutions. Be careful to understand those assumptions in your decision.*

The next section is related to hardware, and addresses the question of where you should run your SSIS-based ETL operations.

Package Execution Location

When you are planning out your SSIS solution architecture, you must consider your package execution strategy. Your objective is to leverage the servers and network bandwidth that can handle the impact load from package execution, but without impacting resources that need primary performance. When it comes to where a package should be executed, there is no absolute answer. However, some general principles can direct one architecture design over another.

Package Storage Location Versus Execution Location

When it comes to running a package, a difference exists between where a package is run and where that package is stored. You can store a package as a file and put it in a file system folder, or you can load a package into the `msdb` system database in SQL Server 2008. Either way, when the package is executed, the storage location is merely where the metadata of that package lives. The package is loaded from that source location through an execution method, and run on the machine where the execution is kicked off. In other words, if you are running a package through the command line or through a job, the package will run on the machine where the command line is called, or the job runs.

Figure 1-9 shows the storage location server on the left and the execution server on the right. The package is executed on the server on the right, even though the package is stored on the server on the left.

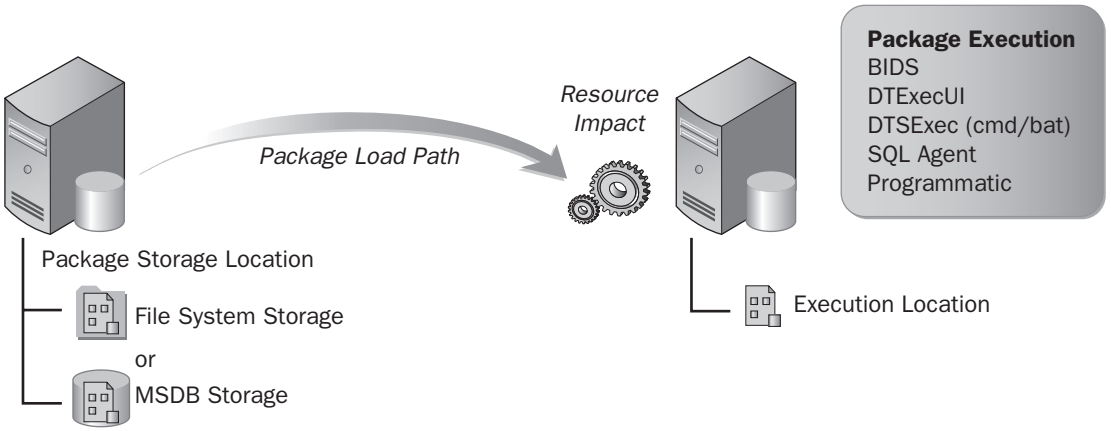

Figure 1-9

Execute SQL Task and Bulk Insert Task Execution

Although a package may be run on any machine with the SSIS service (or, really, that has the SSIS executables and DLLs), this isn't necessarily the place where all the work is being done in the package. For example, if you run an Execute SQL Task in a package and execute the package on a different server than where the Execute SQL Task connection is defined, then the SQL statement is run where the connection is configured, not on the SSIS execution machine. To be sure, the workflow coordination will still be handled on your SSIS execution machine, but the actual SQL code would be run on a different machine.

For the Execute SQL Task and Bulk Insert Task, the SQL code or `BCP` command is executed on the machine that the connection specifies. This is different from the Data Flow Task, which runs on the machine where the package is executed.

Package Execution and the Data Flow

For your packages that have data flows (which is probably most of your packages), you should understand what happens to the data based on where you execute that package (with the embedded data flow). Additionally, understanding where the data flow execution impact will be dictates where you decide to run your packages.

The *data flow impact* on the package execution server involves the resources needed to manage the data buffers, the data conversion requirements as data is imported from sources, the memory involved in the lookup cache, the temporary memory and processor utilization required for the Sort and Aggregate transformations, and so on. Essentially, any transformation logic contained in the data flows is handled on the server where the package is executed.

The following examples are common configurations for where data is sourced, the destination location, and where packages are executed. Obviously, data flows can be varied and complex with multiple sources and destinations, so this simplification provides the framework with single-source locations and single-destination locations.

Packages Executed on the Source or Destination Servers

The most common example is when a package (that contains a data flow) is executed on either the source or destination server, assuming they are separate.

Figure 1-10 shows the data path and impact on the environment when the package is executed on the machine where the source data is located.

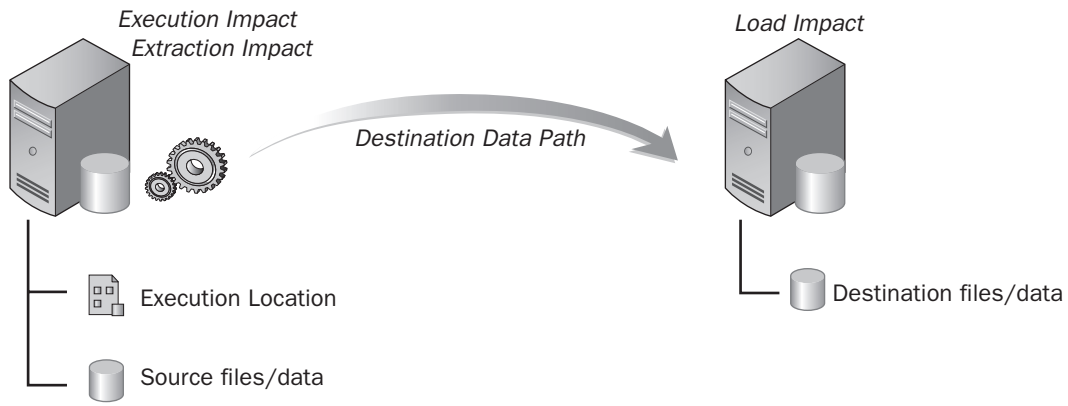

Figure 1-10

The source server will both provide the extracted data and handle the data flow transformation logic, and the destination server will require any data load overhead (such as disk I/O for files, or database inserts or index reorganization).

Following are some of the benefits of this approach:

- There is decreased impact on the destination server, where potential users are querying.
- Data flow buffers are loaded rapidly, given that the location of the source files and package execution is local and involves no network I/O.
- The impact on the destination server is limited, which is useful for destination servers that have 24/7 use, or the SSIS process runs often.

Following are some of the drawbacks of this approach:

- The impact on the source server's resources, which may affect applications and users on the source server
- Potential reduced performance of the data flow destination adapter and the inability to use the SQL Destination adapter, which requires the package be executed on the same server as the package

Destination Server Package Execution

Similar to the impact of running a package on a source server, running a package on the destination server, as Figure 1-11 demonstrates, has similar benefits and drawbacks, just reversed.

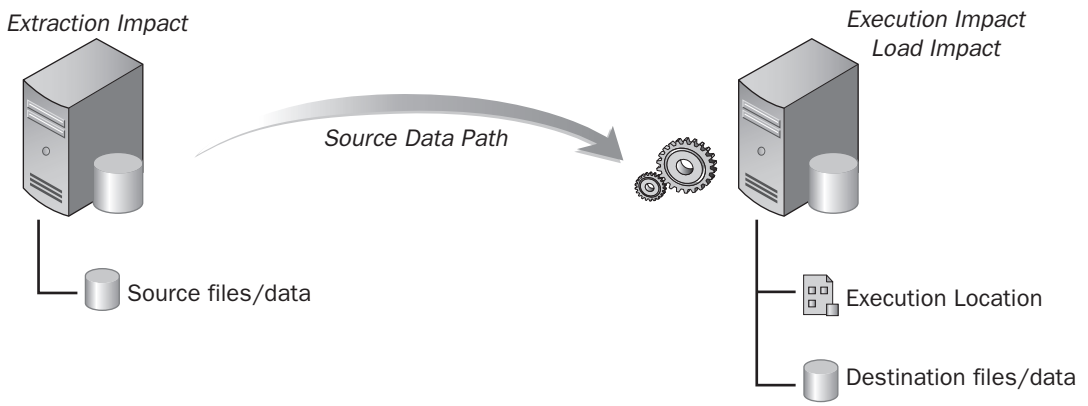

Figure 1-11

Following are some of the benefits of executing a package on a destination server:

- Limited impact on your source server if it is running other critical tasks
- Potential performance benefits for data inserts, especially because the SQL Destination component can now be used
- Licensing consolidation if your destination server is also running SQL Server 2008

One drawback of this approach is that it has a heavy impact on the destination server, which may affect users interacting with the destination server.

This approach is very useful if you have users querying and using the destination during the day, and your SSIS processing requirements can be handled through nightly processes.

Standalone SSIS Servers

An alternative execution option is to run your SSIS packages on a second or third server.

In Figure 1-12, an SSIS package is executed on a second server, and, in this diagram, both the source and destination are on the same machine.

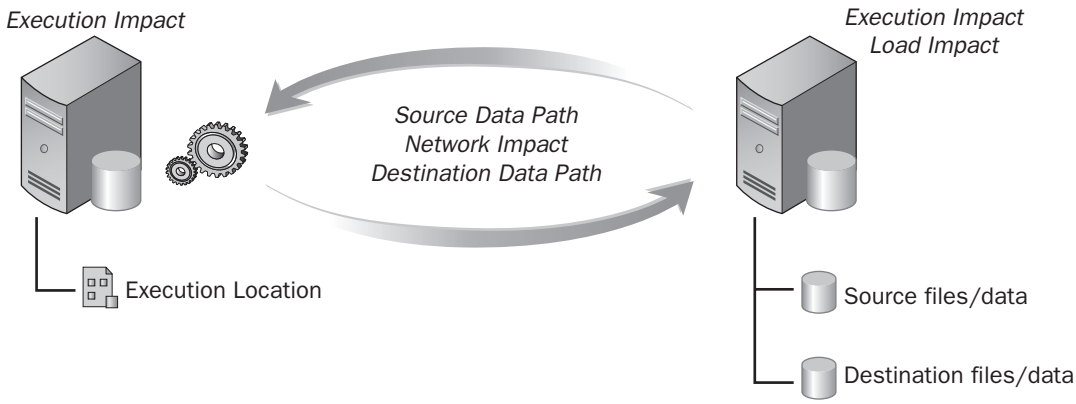

Figure 1-12

As you can see, this scenario is not ideal, because the data would need to travel over the network to the second server, be processed, and then travel back to the same original machine, creating potentially high network utilization, depending on the volume. However, it would reduce the resource impact on the data source and destination server.

Using a standalone SSIS server if your sources and destinations are not on the same physical machines makes more sense. Figure 1-13 highlights this architecture.

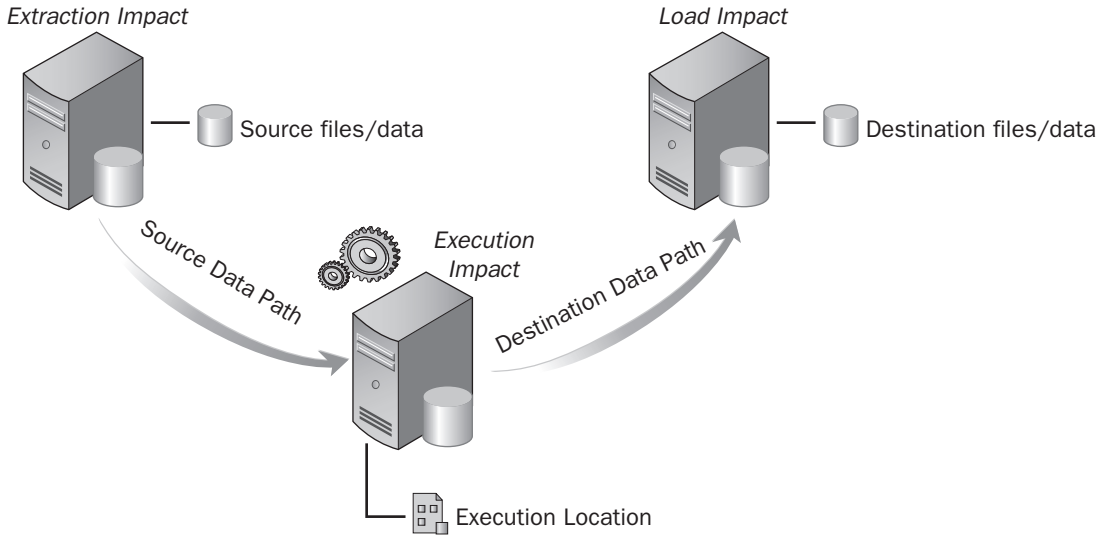

Figure 1-13

In this case, the impact on both the source and destination machines is reduced because the SSIS server would handle the data flow transformation logic. This architecture also provides a viable SSIS application server approach, where the machine can handle all the SSIS processing packages no matter where the data is coming from and going to.

The drawbacks to this approach lie in the capability to optimize the source extraction and destination import, increased network I/O (because the data has to travel over the wire two times), as well as licensing.

Design Review

As you can see, you have a lot to juggle at once when you are planning and building an ETL or data-integration solution. In many cases (such as the infrastructure), all you need to do is set the ball in motion with your IT hardware group. If you are working with a smaller team, you will have more to do, but the discussion in this "Design" section gives you direction to go in, and helps you stay on top of your project. Sometimes you may feel like you are juggling several balls at once, but you'll be better off starting early, rather than trying to juggle ten balls in the middle of the project!

Solution

Now it is time to focus on the SSIS ETL solution. The project is in motion, expectations are set with the stakeholders, and you have laid the foundation to a successful project.

The next step is about designing and implementing your SSIS solution. Just when you think that you have a handle on things, you now have to dive into the details of data and processes! This section launches you into your SSIS design . . .not by providing all the steps in these next few pages, but by giving you the driving directions on where to find your design and solution answers in the rest of this book.

The chapters in this book flow in the natural progression that you may go through when designing your SSIS solution. The next couple chapters provide you with the underlying SSIS support structure for your solution — the storage, deployment, and management framework.

Next, as you delve into the data, you will be dealing with source data, whether in files or extracted from a relational database management system (RDBMS) and often requiring a data-cleansing process. Chapters 4–6 cover files, data extraction, and cleansing.

If your SSIS solution involves data warehouse ETL (which involves dimension and fact table loading, and often cube processing) Chapters 7–9 are for you.

The final chapters address advanced package scalability and availability, advanced scripting for those really complex scenarios, and performance tuning and design best practices.

Setting the Stage: Management and Deployment

One of the very first things you must design is an SSIS package template that integrates with a management and auditing environment. You must do this upfront, because retrofitting your logging and auditing while your packages are being designed is very difficult.

Chapter 2 examines building an SSIS management framework for this purpose. A management framework is about knowing the what, when, and why of when a package executes. Knowing these items is critical for SSIS administration and troubleshooting. Chapter 2 considers how to approach creating package templates, supporting auditing structures, and centralized configurations.

The next design task is also related to management. It is defining your package deployment strategy and your package storage model where you save your packages. Chapter 3 looks at all the surrounding issues and factors involved in choosing the right model for your situation.

Deciding on a deployment and package storage model (the file system or in the `msdb` system database) is important early on because, as the number and design complexity of your SSIS packages grows (in an organization), so do the challenges related to package deployment and management.

Source Data: Files, Tables, and Data Cleansing

Every ETL or data-integration project involves data. And this data must come from somewhere, either from a file provided or a table (or query). Therefore, you are going to deal with source data in some aspect.

Chapter 4 discusses how to deal with files. Many of the data-integration and processing projects involve working with data files of many kinds, such as delimited files, binary files, image files, Excel files, or XML files. These file types and other types may need to be moved around, modified, copied, or even deleted, and their contents may need to be imported. Chapter 4 walks you through the methods and practices using SSIS to handle file management.

Your project may not involve files, but even if it does, you will more than likely have to extract data from a source database. Chapter 5 examines data extraction. SSIS solutions often have a data-extraction component, which may involve connecting to relational systems like Oracle, DB2, Sybase, MySQL, or TeraData. Chapter 5 considers the best practices for extracting data from various sources, and addresses

different extraction requirements (such as optimization, incremental or targeted extraction, change data capture, or data staging), and also addresses common challenges from the various source types, 32-bit or 64-bit platform.

If someone ever tells you that his or her source data is perfect, he or she is lying. Don't believe it! The fact is that data sources are rarely pristine, and often require handling data anomalies, missing data, typographical errors, and just plain bad data. Chapter 6 delves into the practical design steps to handle data cleansing in SSIS using the fuzzy logic, text parsing, and scripting components for data cleansing and text mining. Chapter 6 also demonstrates the use of the SSIS Data Profile Task to better understand the source data that you are dealing with.

Data Warehouse ETL and Cube Processing

The "Design" section of this chapter already reviewed the overall ETL architecture of a data warehouse and BI system. As a background to Chapter 7 (which discusses dimension tables) and Chapter 8 (which discusses fact table ETL), databases designed for data warehousing are created in a structure called a *dimensional model*, which involves two types of tables:

- *Dimension tables* hold informational data or attributes that describe entities.
- *Fact tables* capture metrics or numeric data that describe quantities, levels, sales, or other statistics.

A data warehouse involves both dimension tables and fact tables. Therefore, your ETL processes need to handle the transformation and loading of dimension tables and fact tables.

Chapter 7 focuses on loading data into data warehouse dimension tables, which requires handling attribute changes, managing business keys, and dealing with surrogate keys. Chapter 7 dives into the best practices for loading different dimension table types, and examines how to optimize the SSIS process to scale to large and complex dimension tables.

Chapter 8 focuses on loading data warehouse fact tables, which often contain millions or billions of rows. Loading data into a fact table requires designing an SSIS package that can scale and also handle the various loading scenarios (such as table partitions, large record updates, and missing keys). Chapter 8 considers the design practices for loading the different types of fact tables.

A data warehouse is often accompanied by a set of cubes. A Microsoft cubing engine, which is a part of SQL Server, is called SQL Server Analysis Services (SSAS). Data warehouse or business intelligence solutions that use SSAS need a data-processing architecture that loads data from the data warehouse or mart into SSAS. Chapter 9 focuses on how to use SSIS to load SSAS data cube structures through the out-of-the-box SSAS processing components, as well as through scripting and the command-line tools. Also included in Chapter 9 is a discussion on how to manage SSAS partitions, and how to deal with incremental data updates.

Advanced ETL: Scripting, High Availability, and Performance

The final three chapters of this book deal with the advanced topics of scripting, high availability, and scaling your package. Chapter 10, for example, dives into how to build a scale-out SSIS solution for high-availability solutions.

One of the most powerful tools within SSIS is the scripting engine. Often, the out-of-the box tasks and transformations cannot handle a unique situation effectively. However, the Script Task and Script component can often be the perfect solution to a complicated problem. Chapter 11 demonstrates examples (both control flow and data flow) where SSIS scripting can solve challenging requirements effectively.

Finally, Chapter 12 addresses the problem of performance and scalability. Data volumes and complexity can sometimes inhibit SSIS package performance. In fact, even SSIS packages designed for simpler requirements can have performance challenges, and if you are dealing with tight data-processing windows, or you want to limit the impact of packages, then scaling and tuning your packages is important. Chapter 12 delves into the best practices for tuning, as well as isolating and resolving performance problems.

Summary

Enterprise environments are often difficult to work in, given the complexities of system dependencies. Even if your organization's data nervous system is limited, or your SSIS project is only a small part of your bigger environment, you must carefully plan how your solution integrates into it. In fact, you should use your ETL project as an opportunity to demonstrate how to build a solution the right way, from the infrastructure, to the processes, to the package design, and management. Approach your project with an overall architecture in mind, be diligent about documenting data elements, plan for the right infrastructure and process execution strategy, and consider solution and data growth as the project is expanded, or more data is loaded or integrated.

When you launch your project in the right direction, the very next thing you should do is implement an SSIS management structure. Chapter 2 examines building an ETL management framework for your packages.